PCT/GB 2004 / 0 0 1 1 6 8

The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

REC'D 0 4 MAY 2004
WIPO PCT

PRIORITY DOCUMENT

SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed [illegible]

Dated 2 April 2004

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
K
16 APR 2003
LONDON

The Patent Office

1/77

17APR03 E800889-1 C01570
P01/7700 0.00-0308840.8

The Patent Office
Cardiff Road
Newport
Gwent NP10 8QQ

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

1.	Your reference	A30353
2.	Patent application number *(The Patent Office will fill in this part)*	0308840.8 16 APR 2003
3.	Full name, address and postcode of the or of each applicant *(underline all surnames)*	BRITISH TELECOMMUNICATIONS public limited company 81 NEWGATE STREET LONDON, EC1A 7AJ, England Registered in England: 1800000
	Patents ADP number *(if you know it)*	1867002
	If the applicant is a corporate body, give the country/state of its incorporation	UNITED KINGDOM ✓
4.	Title of the invention	FLEXIBLE MULTI-AGENT SYSTEM ARCHITECTURE
5.	Name of your agent *(if you have one)*	
	"Address for Service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	BT GROUP LEGAL INTELLECTUAL PROPERTY DEPARTMENT HOLBORN CENTRE 120 HOLBORN LONDON, EC1N 2TE
	Patents ADP number *(if you know it)*	~~1867001~~ 8591919001

		Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*
6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number			

		Number of earlier application	Date of filing *(day/month/year)*
7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application		

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
(See note (d))

YES

9. Enter the number of sheets for any of the fo[illegible]ing items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	32 ✓
Claim(s)	5 ✓
Abstract	1 ✓
Drawing(s)	11 + 11 [illegible]

10. If you are also filing any of the following, state how may against each item

Priority Documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature(s) [illegible] Date: 16 April 2003

NASH, Roger William, Authorised Signatory

12. Name and daytime telephone number of person to contact in the United Kingdom: Rod Hillen 020 7492 8140

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

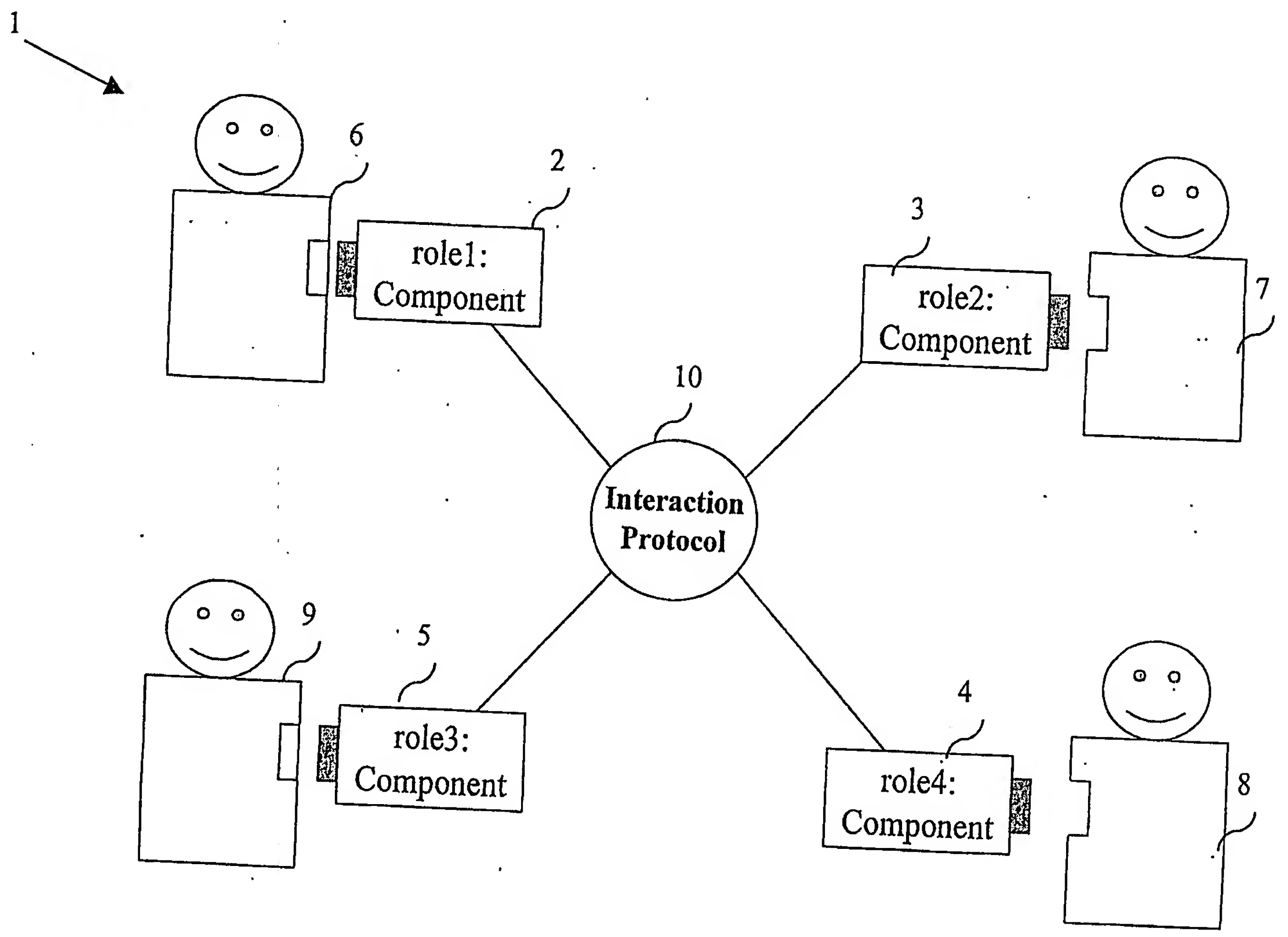

1
6
2
role1: Component
3
role2: Component
7
10
Interaction Protocol
9
5
role3: Component
4
role4: Component
8

FIG. 1A

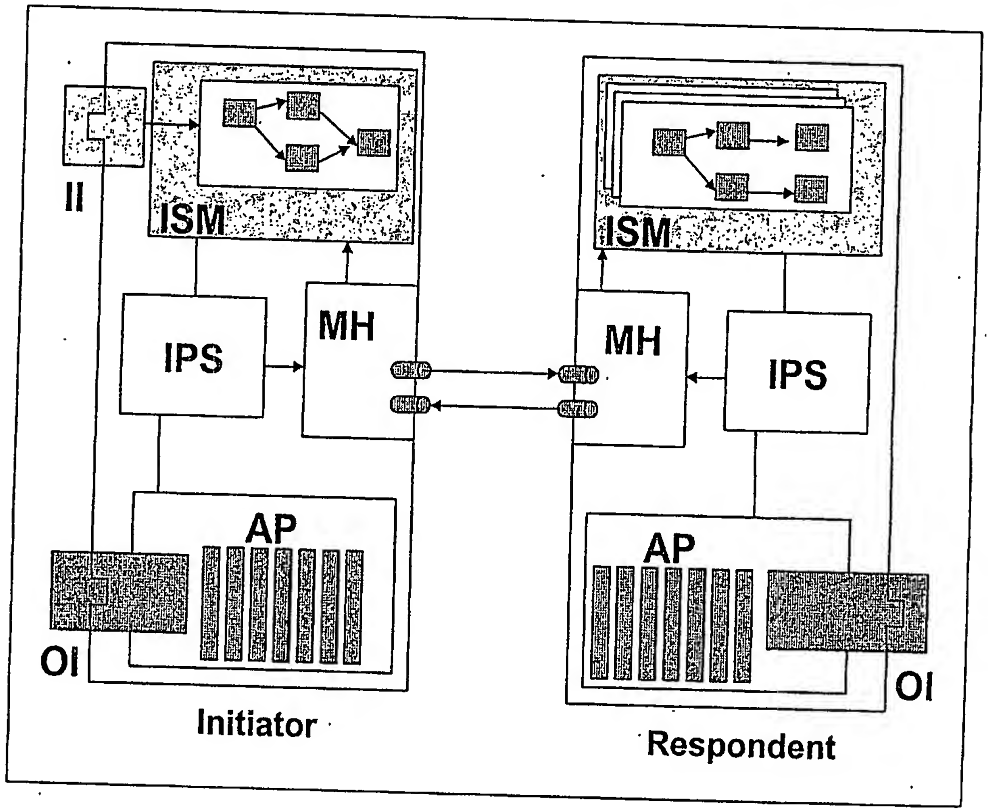


FIG. 1B

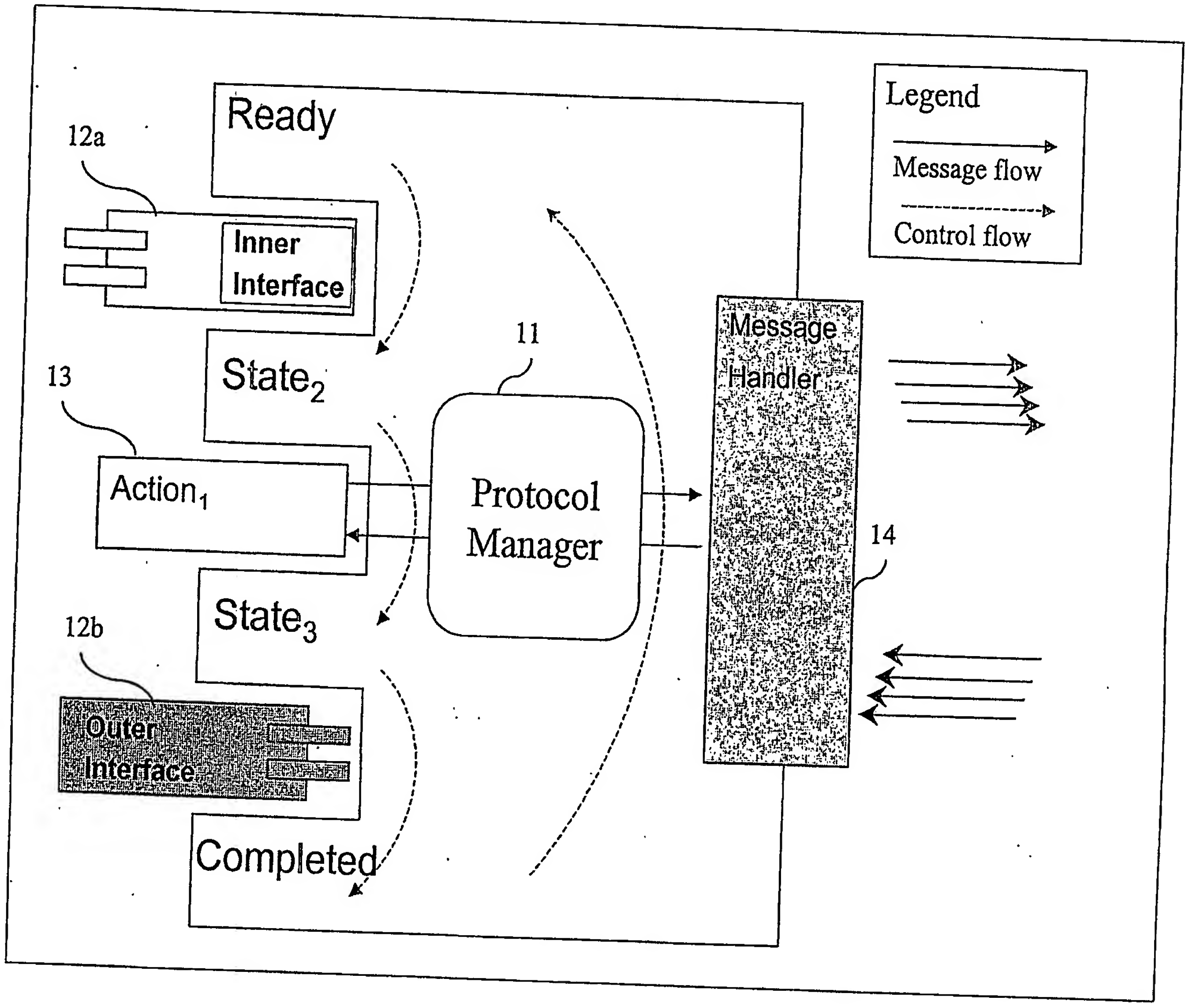

Legend
Message flow
Control flow
Ready
12a
Inner Interface
State$_2$
13
Action$_1$
State$_3$
12b
Outer Interface
Completed
11
Protocol Manager
Message Handler
14

FIG. 2

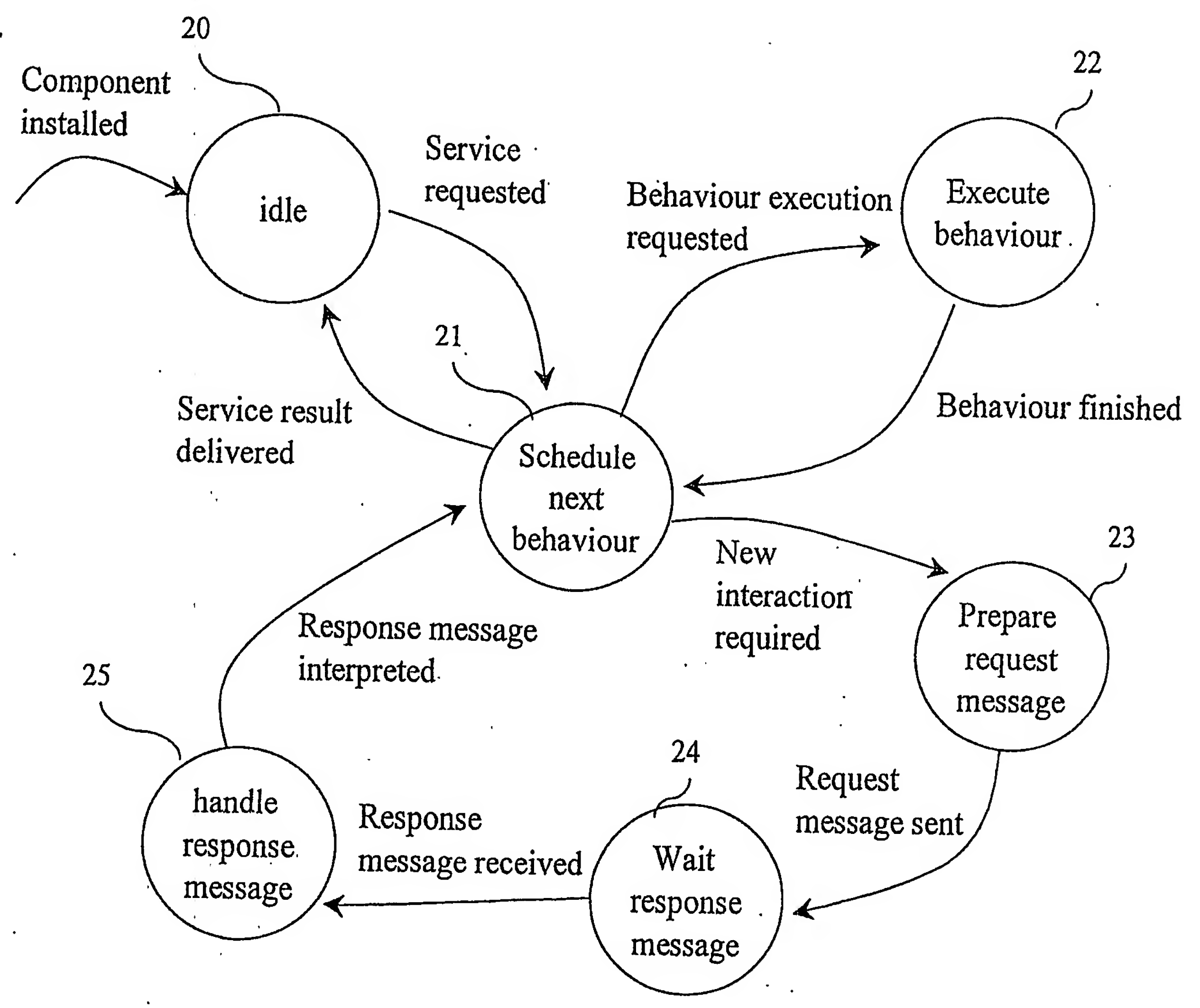

20
Component installed
idle
Service requested
Service result delivered
21
Schedule next behaviour
Behaviour execution requested
22
Execute behaviour
Behaviour finished
New interaction required
23
Prepare request message
Request message sent
24
Wait response message
Response message received
25
handle response message
Response message interpreted

FIG. 3A

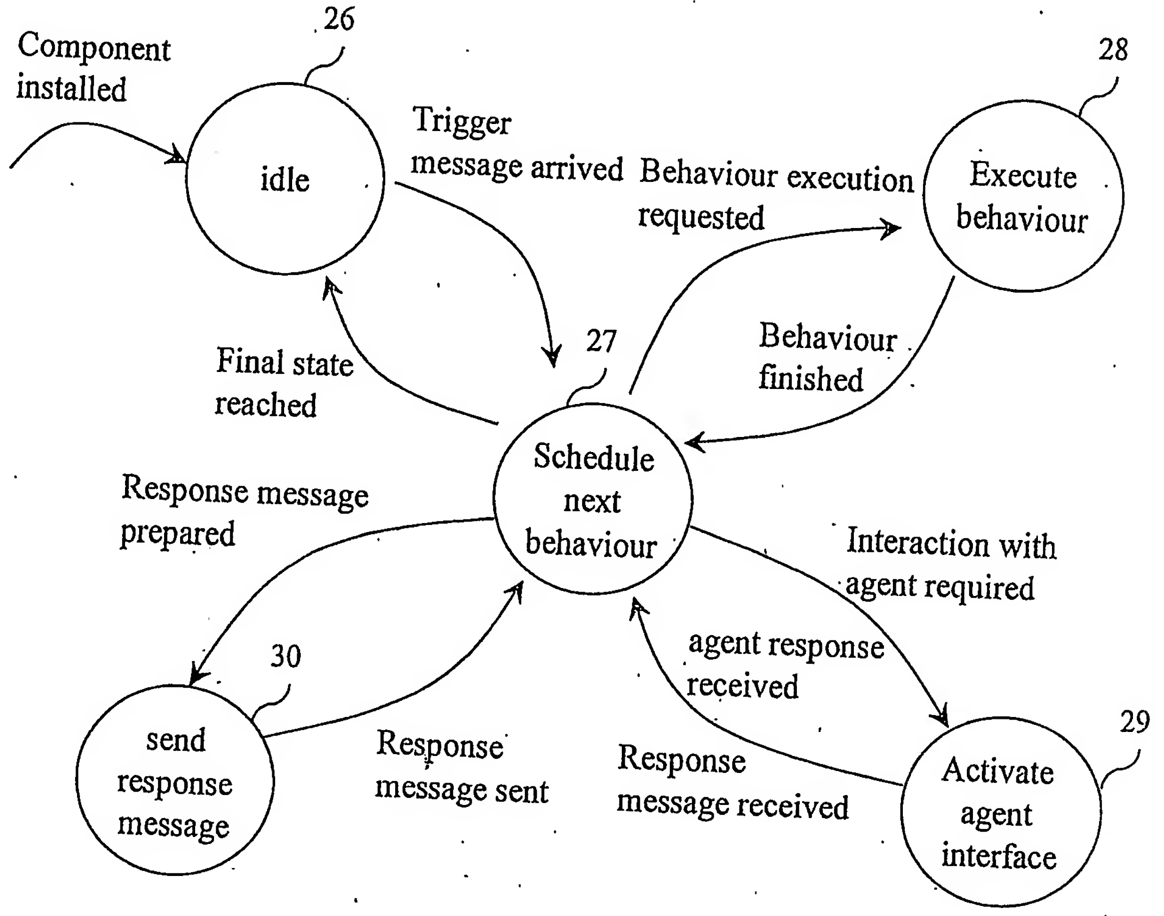

Component installed
26
idle
Trigger message arrived
Behaviour execution requested
28
Execute behaviour
Final state reached
27
Behaviour finished
Schedule next behaviour
Response message prepared
Interaction with agent required
agent response received
30
send response message
Response message sent
Response message received
29
Activate agent interface

FIG. 3B

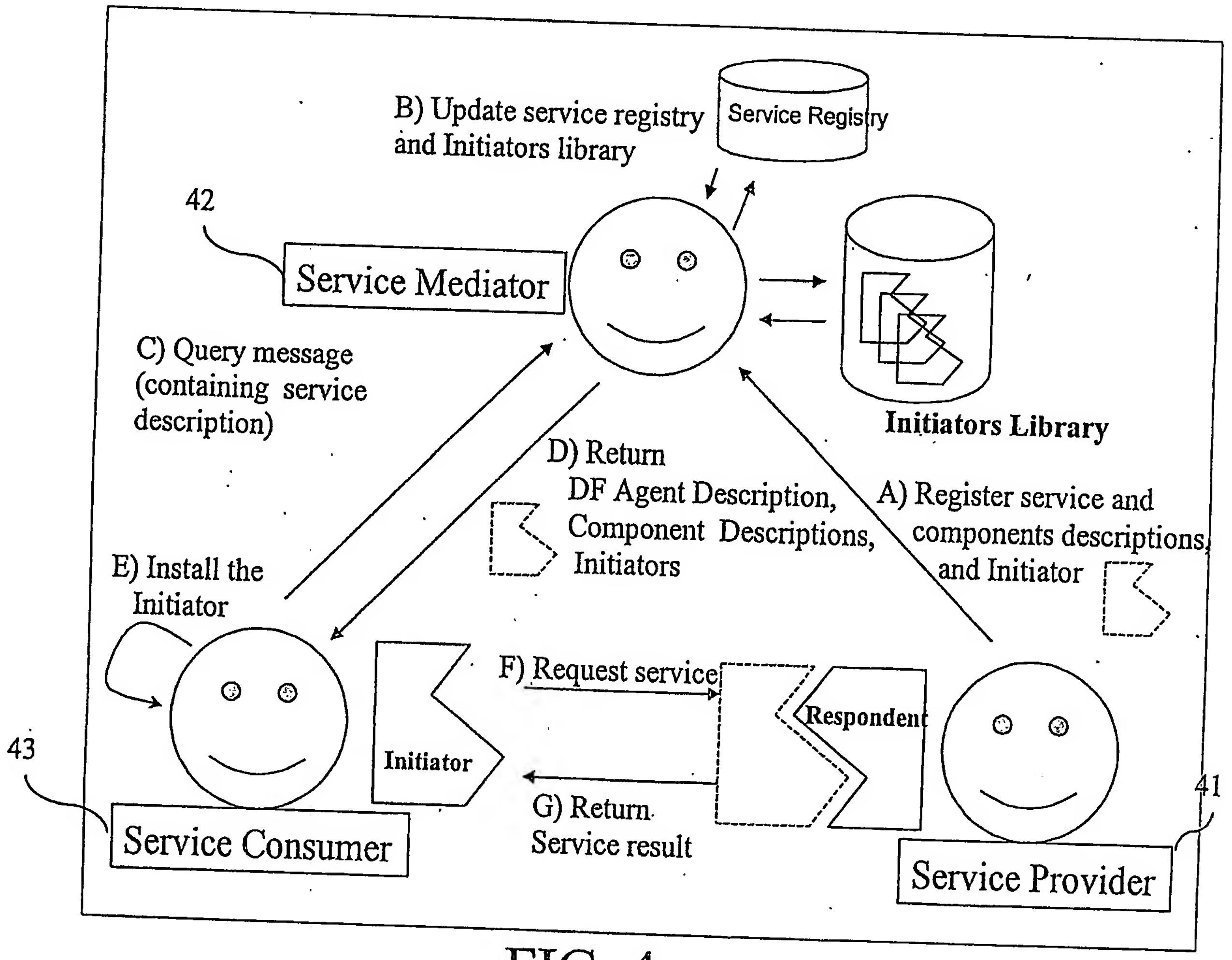

B) Update service registry
and Initiators library
Service Registry
42
Service Mediator
Initiators Library
C) Query message
(containing service
description)
D) Return
DF Agent Description,
Component Descriptions,
Initiators
A) Register service and
components descriptions,
and Initiator
E) Install the
Initiator
43
Service Consumer
Initiator
F) Request service
G) Return
Service result
Respondent
41
Service Provider

FIG. 4

7/11

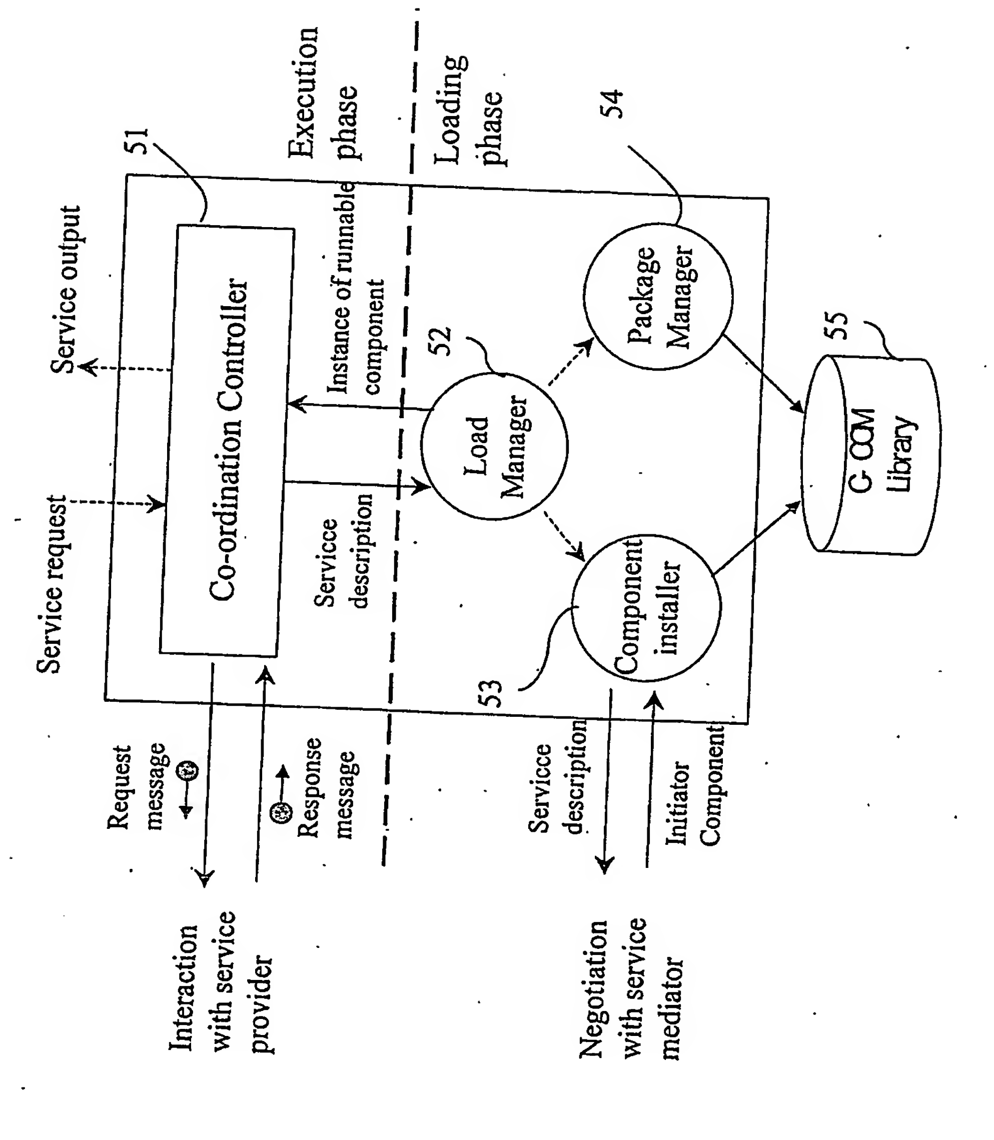

Service request
Service output
Co-ordination Controller
51
Execution phase
Loading phase
Instance of runnable component
Servicce description
52
Load Manager
54
Package Manager
55
G CCM Library
Component installer
53
Request message
Response message
Interaction with service provider
Servicce description
Initiator Component
Negotiation with service mediator

FIG 5

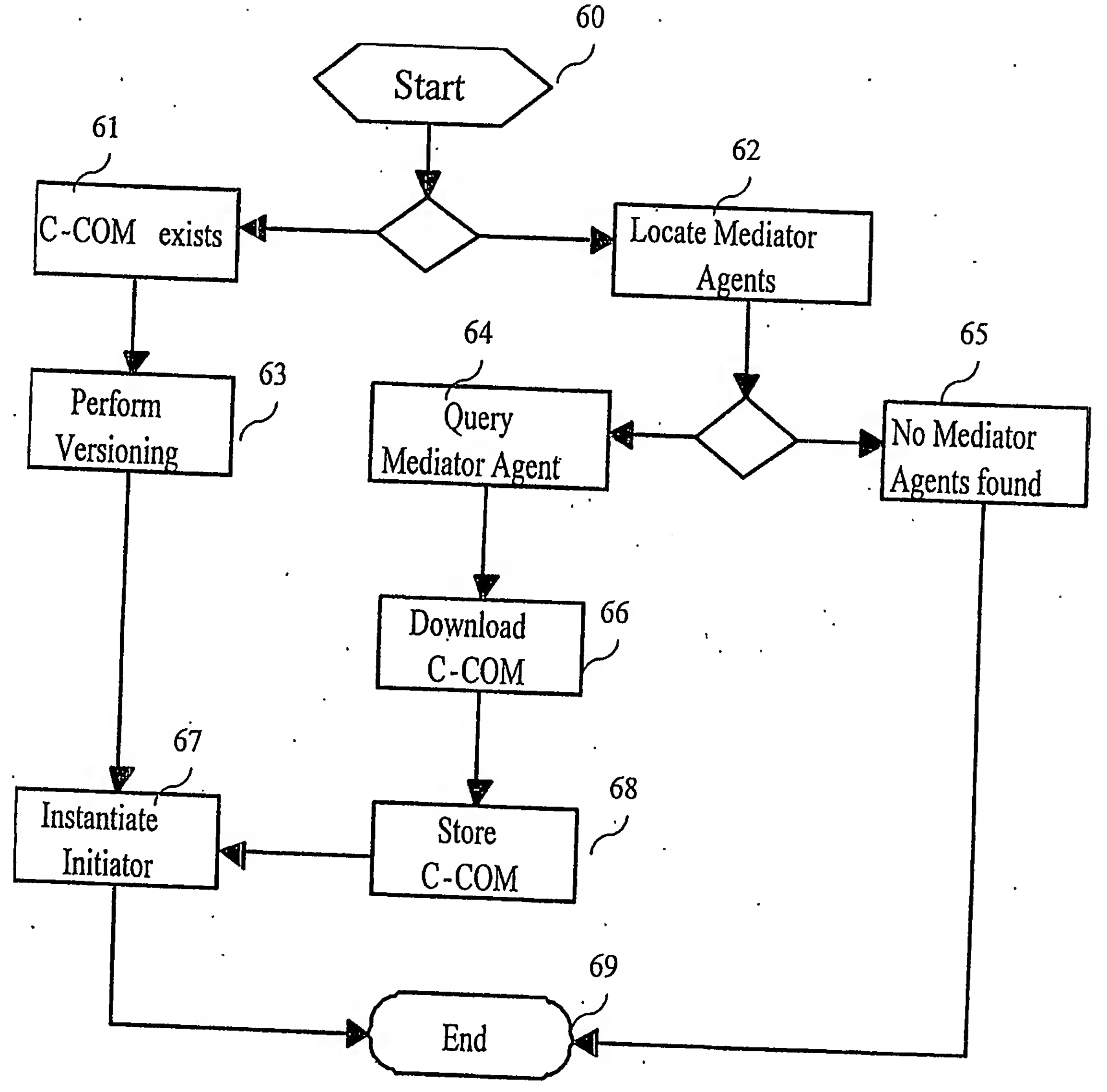
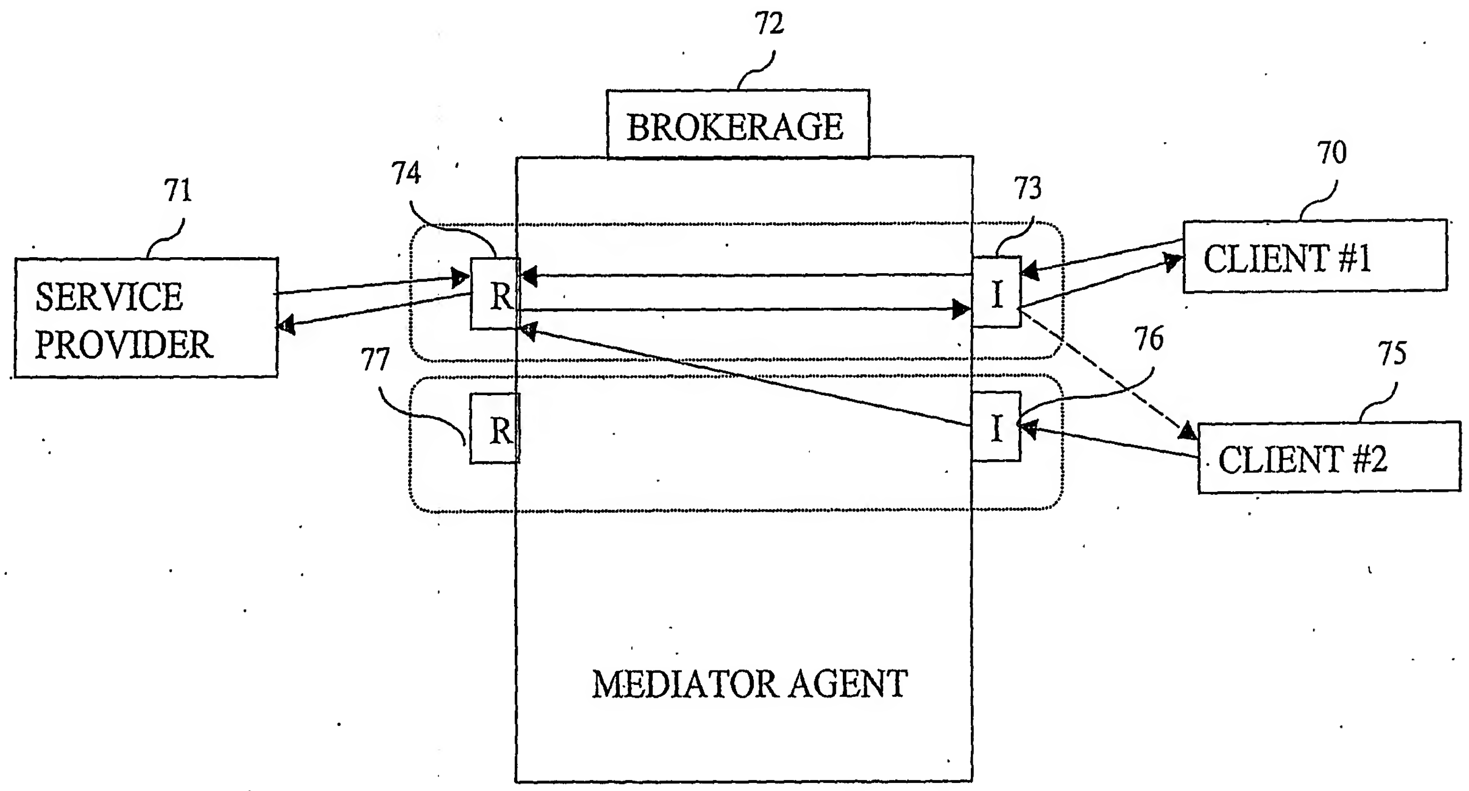

60
Start
61
C-COM exists
62
Locate Mediator Agents
63
Perform Versioning
64
Query Mediator Agent
65
No Mediator Agents found
66
Download C-COM
67
Instantiate Initiator
68
Store C-COM
69
End

FIG 6

9/11



72
BROKERAGE
71
SERVICE PROVIDER
74
R
77
R
73
I
76
I
70
CLIENT #1
75
CLIENT #2
MEDIATOR AGENT

FIG. 7

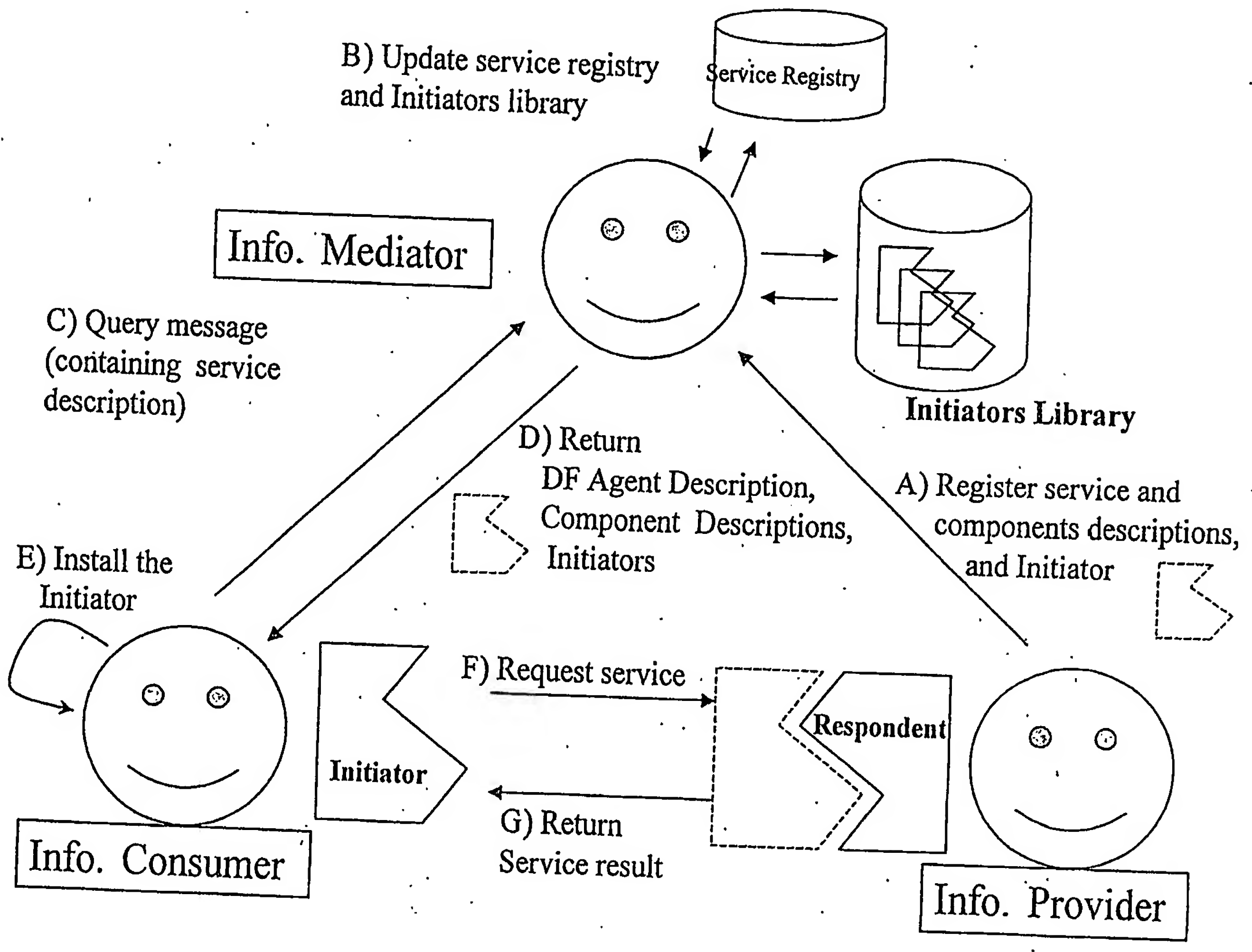

B) Update service registry and Initiators library
Service Registry
Info. Mediator
Initiators Library
C) Query message (containing service description)
D) Return DF Agent Description, Component Descriptions, Initiators
A) Register service and components descriptions, and Initiator
E) Install the Initiator
F) Request service
Respondent
Initiator
G) Return Service result
Info. Consumer
Info. Provider

FIG. 8A

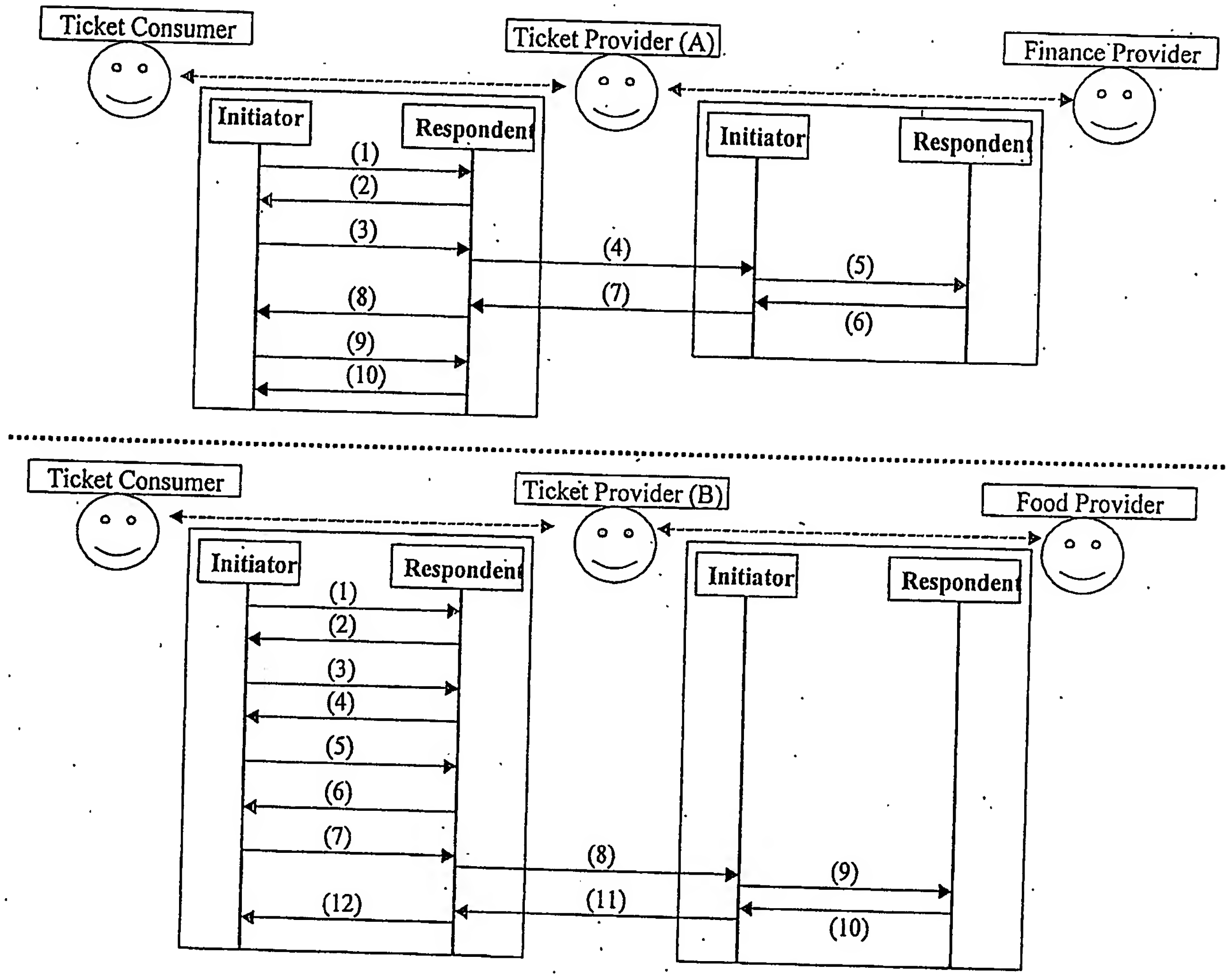

Ticket Consumer
Ticket Provider (A)
Finance Provider
Initiator
Respondent
Initiator
Respondent
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
Ticket Consumer
Ticket Provider (B)
Food Provider
Initiator
Respondent
Initiator
Respondent
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)

FIG. 8B

Flexible Multi-Agent System Architecture

The present invention relates to a multi-agent system (MAS) architecture, in particular to a multi-agent system architecture which is suitable for Open Electronic Commerce. The invention further relates to a method of, and a mediator agent for, providing generic role components to other agents in the MAS.

Software agent technology is widely used in a variety of applications ranging from comparatively small systems such as personalised email filters to large, complex, and mission-critical systems such as air-traffic control. Multi-Agent Systems (MASs) are designed and implemented using multiple software agents that interact via messages to achieve a goal. MASs are used in the field of information service provision where information service providers are highly competitive and it is very advantageous if they can differentiate their products by providing new kinds of interactions amongst information customers.

In a MAS, each agent has incomplete information or a limited capability for solving a problem. Therefore an agent must interact with other agents autonomously. A MAS can be differentiated from existing distributed object systems in that each agent autonomously detects and solves a problem using its reasoning facility minimizing the human user's intervention.

One of the main MAS principles concerns the separation of service provision and service requests amongst the distributed agents. If one agent cannot perform a task, it adopts the role of a client agent and requests assistance from another agent (acting in the role of server agent) which satisfies the request by executing the required service.

Currently, interactions among multiple agents are affected by certain limitations of known MASs. These limitations include interoperability issues among heterogeneous agents, the semantics of the agent communication language (ACL) used which specifies the standard agent message structures, the allocation of tasks among participant agents, and the building of conversation policies (or interaction protocols

(IPs)) etc. (For more details see Mamadou T. Kone, Akira Shimazu and Tatsuo Nakajima, "The state of the art in agent communication languages", in Knowledge and Information Systems (2000) 2: 259-284; and Jennings, N. R., Sycara, K., and Wooldridge, M., "A roadmap of agent research and development", Autonomous Agents and Multi-Agent Systems, 1, 275-306, 1998.

Interoperability is mainly concerned with making different agents communicate with each other by using standard ACL and interaction protocols etc. Interoperability involves several areas of research such as ontology, content language, and ACL. Ontologies provide common representations of knowledge for specific domains where agent communication is made. Content languages are standard expressions of domain-independent knowledge that are used together with ontologies to specify the content part of agent messages.

Whilst interoperability is not an issue when all agents on the same platform use a predefined language, ontology, and interaction protocols to compose an ACL, this situation is unrealistic in an electronic commerce environment where new agents are dynamically introduced with new services.

FIPA (The Foundation for Intelligent Physical Agents) aims to produce standards for the interoperation of heterogeneous software agents. FIPA has developed the FIPA Abstract Architecture Specification which specifies the standard architecture that heterogeneous agent platforms should comply with to be able to communicate each other.

According to the FIPA Abstract Architecture Specification (for more details see FIPA Specifications, Foundation for Intelligent Physical Agents, 2000, http://www.fipa.org/repository/index.html), a server agent should register its services with a directory facilitator (DF) and client agents should contact the DF to find an appropriate server agent that provides the required services. The client agent creates a service description that contains the service name, type, protocol, ontology, and content language to be used for the service and uses the service description to query the DF to find suitable server agents. However, this has the limitation that a client agent is only able to request services which are already known to it (for example, see Steven Wilmott, Jonathan Dale, Bernard Burg, Patricia Charton and Paul O'Brien, "Agentcities: A Worldwide Open Agent Network", Agentlink News, No. 8, Nov. 2001, available at http://www.AgentLink.org/newsletter/8/AL-8.pdf, for more details). Moreover, FIPA addresses the issue of providing a mechanism that allows interoperability between agents in a variety of heterogeneous platforms by using a standard message structure or content language, ontology, and interaction protocol (these standards can also be used within the same platform). However, the standards do not specify how existing agents can handle messages with any combination of new content language, ontology, or interaction protocol even if they reside on the same platform.

Another issue in relation to interoperability is the conversation policy used for the multi-agent interaction. Conversation policies, also called interaction protocols, are predefined sequences of agent messages that guide and constrain agent communications for specific objectives. They are essential in complicated agent conversations that involve a lot of messages and many possible branches of logic and have been the subject of research by several parties, for example, see the earlier reference by Mamandou as well as Ren'ee Elio and Afsaneh Haddadi, "On abstract task models and conversation policies", in Working Notes of the Workshop on Specifying and Implementing Conversation Policies, pages 89--98, Seattle, Washington, May 1999; M. Greaves, H. Holmback, and J. Bradshaw, "What is a conversation policy? ", in Proc. *The Workshop on Specifying and Implementing Conversation Policies*, Seattle, Washington, May 1999, pp. 118-131; Scott A. Moore, " On conversation policies and the need for exceptions" , in Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Seattle, Washington, May 1999; and Jeremy Pitt and Abe Mamdani, "Communication protocols in multi-agent systems", In Working Notes of the Workshop on Specifying and Implementing Conversation Policies, pages 39--48, Seattle, Washington, May 1999.

Without a conversational policy, individual agents in each communication step may face difficulties in determining how to communicate with each other by choosing which message type to use, based on their own understanding and implementation of the ACL semantics. It therefore is generally advisable to use conversation policies in all non-trivial conversations.

One of the limitations of known MASs is that agents use only a set of known or standard conversation policies and cannot handle ad-hoc conversation policies without re-implementation. The need for ad-hoc conversation policies is clear in information-centric agents in e-markets that are characterized by their focus on collecting, processing, analysing, and monitoring information.

Information-centric agents, also referred to herein simply as information agents, can be defined as a class of autonomous agents that are closely related with information sources (for more details see K. Decker, A. Pannu, K. Sycara, and M. Williamson, "Designing behaviors for information agents", in Proc. *The First International Conference on Autonomous Agents (AGENTS-97)*, Feb. 1997, pp. 404-412). For these reasons, conversations with information agents can be dependent on the nature of information sources. As information sources range from legacy systems to web sites, etc, it is not appropriate to assume that in future they will be able to be supported by a few standard conversation policies. Even now, it has been proposed that existing conversation policies are not extensive enough to support all applications and systems (see, for example, the above reference by Moore) and new policies will need to be implemented and present policies upgraded in future. A conversation policy handshaking mechanism to allow agents to exchange ad-hoc conversation policies and interact after interpretation of the new conversation policy on the fly is already known in the art from Hyung Jun Ahn, Habin Lee, Hongsoon Yim, Sung Joo Park, "Handshaking Mechanism for Conversation Policy Agreements in Dynamic Agent Environment," *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002).* However, this ad-hoc interaction protocol is not able to handle messages with unknown language and ontology.

A service component concept for MAS has been adopted in the European 5th Framework project LEAP (Lightweight Extensible Agent Platform) (see LEAP Project website, http://leap.crm-paris.com). In the LEAP project, a Generic Service

Component (GSC) construct was designed, and 22 specializations of it, covering three application areas (that is, knowledge management, travel management, and decentralised work co-ordination management) were produced (a subset of which were actually implemented) to be used in a wide variety of applications. However, whilst the main objective of a LEAP GSC is to provide generic service components which can be reused in similar applications, the GSC construct developed in the LEAP project is a static library that is used when agents are developed, and which must be present when they are launched on an agent platform. Consequently, the generic service components proposed by LEAP cannot be dynamically installed on to agents which are already running.

In electronic commerce (eCommerce), MASs are considered to play a major role (see for example, Maes, Pattie, Guttman, R. H., and Moukas, A. G. "Agents That Buy and Sell", Communication of the ACM, Vol. 42, No. 3, pp.81-91, 1999). In eCommerce, autonomous agents can act on behalf of their human users to buy or sell products or services and the openness and flexibility of the MAS will affect its success. The eCommerce environment is dynamic in that new consumers and merchants may appear, new products and services can be introduced frequently, and sometimes the standard languages and transaction rules can be changed. As a result, any MAS implemented in an eCommerce environment needs to be flexible enough to adapt to frequent changes in eMarket settings. More specifically, a MAS for eCommerce should enable agents to participate, disengage, and/or transact with each other using new business rules (subject to some minimum constraints) whenever these new rules arise.

In order for a known MAS to become sufficiently open and flexible for the eCommerce environment, all participating agents would be required to use the same content languages, ontologies, and interaction protocols in the messages they exchange. This is unrealistic to achieve using known MASs as this prerequisite makes it impossible for agents providing new services based on a new interaction protocol, ontology, or content language to participate in any existing MAS-based markets. To accommodate such an agent, the MAS would have to be re-engineered to allow existing agents to use the new content language, ontology, or interaction protocol to request and receive service from the new agent.

The present invention seeks to obviate and/or mitigate the above problems and disadvantages known in the art by providing a multi-agent system architecture which provides sufficiently flexibility to support an evolving eCommerce environment.

A first aspect of the invention seeks to provide a service component arranged in use to enable a client agent to interact with a server agent when requesting a service, the service component comprising: a plurality of role components arranged in use to perform a service interaction between the client agent and the server agent, the role components being loaded onto said client and server agents as appropriate for the interaction and when loaded arranged to provide the client and server agents with information on the interaction requirements to enable the requested service to be provided.

Preferably, the Initiator role components are provided by a service provider agent to a service consumer agent.

Preferably, the role components are attached with a component description, the component description including details of the minimum client platform capability of the client agent and the interfaces used by the client agent to interact with the role component.
Preferably, the Initiator role component can control its state and can be reused for multiple requests.

Preferably, the role components are distributed by a mediator agent.

Preferably, the mediator agent provides the role components dynamically to the client and server agents.

More preferably, the mediator agent identifies a suitable role component using a service description and component description of the role component.

Preferably, one of said plurality of role components is an Initiator role component provided dynamically to the client agent whilst the client agent is running.

Preferably, one of said plurality of role components is a Respondent role component provided dynamically to the server agent whilst the server agent is running.

A second aspect of the invention relates to a service component arranged to enable a client agent to interact with a server agent when requesting a service, the service component comprising: a plurality of role components arranged to perform a service interaction between the client agent and the server agent, the role components providing the client and server agents with information on the interaction requirements to enable the requested service to be provided, wherein the service component is generic to the client and server agents and is dynamically installed into at least one of the client and server agents when these agents are already running.

A third aspect of the invention relates to a multi-agent service architecture having a service component arranged to enable a client agent to request a service from a service agent, the architecture including: a mediator agent arranged to provide a role component to the client agent, wherein once the role component is loaded on the client agent, the client agent is provided with information which enables the service to be provided by the service agent.

A fourth aspect of the invention relates to a method of providing a user with access on demand to a remote service, the method comprising the steps of: generating a client agent for the user to request the service from a server agent; providing the client agent with at least one service component arranged to modify the client agent to enable the client agent to interact with the server agent when requesting the service; forwarding the modified client agent to the broker to enable the server agent and modified client agent to interact; and responding to the client agent's request to provide the requested service, wherein the service component provided comprises:
a plurality of role components arranged to perform service interactions between the client agent and the server agent, the role components providing the client and server

agents with information on the interaction requirements to enable the requested service to be provided.

A fifth aspect of the invention relates to a method of enabling a software agent to participate in an inter-agent interaction in a MAS architecture, the method comprising the steps of: determining at least one of a plurality of role components for use by a service component of said software agents required for participation in the inter-agent interaction; identifying a mediator agent in the MAS which is capable of providing at least one role component required by the software agent for participation in the inter-agent interaction, the mediator being identified by means of a service component description as having a suitable role component for the service component; dynamically installing the role component provided by the mediator agent on the software agent; and loading the role component on the software agent to enable the software agent to participate in the inter-agent interaction.

A sixth aspect of the invention relates to an agent internal architecture for dynamically installing and executing role components, the architecture comprising:

a Co-ordinator controller, a Load manager, a Component installer, and a Package manager.

A seventh aspect of the invention relates to a method of enabling a software agent to manage downloaded role components, the method comprising the steps of: determining whether there are any components downloaded already in local component storage; performing version checking for downloaded Initiator role components if there exist any Initiator role components; locating the Mediator agent and downloading any Initiator role components from the Mediator Agent; packaging the downloaded Initiator role components into local component storage; and instantiating the downloaded Initiator role components.

An eighth aspect of the invention relates to a computer product comprising a suite of one or more computer programs provided on a computer readable data carrier, the one or more computer programs being arranged to implement any one of the method aspects of the invention.

A ninth aspect of the invention relates to a signal conveying a suite of one or more computer programs over a communications network, the suite of one or more computer programs being arranged when executable to implement any one of the method aspects of the invention.

It will be appreciated by those skilled in the art that the invention can be implemented in any appropriate combination of software and hardware, and that invention can also be implemented by a synergy of software and hardware, for example, when a computer software product comprising one or more computer programs arranged to implement any one of the method aspects of the invention is run on a computer.

The standardization of languages and interaction protocols used in messages during the interaction among information agents in electronic commerce frustrates the needs of service providers as they attempt to differentiate their services. Advantageously, the multi-agent system architecture of the invention uses conversational components as the main tool for interactions amongst participating agents. Advantageously, this allows an information consumer agent to interact with an information provider agent to supply an information service via an unknown language or interaction protocol.

Advantageously, the architecture of the invention enables agents residing on the same agent platform to interact using a new language, ontology, or interaction protocol. Advantageously, an ad-hoc interaction protocol is provided by the architecture which can handle messages with unknown language and ontology.

Advantageously, the generic service components of the invention can be dynamically installed into agents which are already running.

The features of the invention as defined above or by the dependent claims may be combined in any appropriate manner with any appropriate aspects of the invention as apparent to those skilled in the art.

The preferred embodiments of the invention will now be described with reference to the accompanying drawings, which are by way of example only and in which:

Figure 1A shows the internal architecture of a conversational service component according to the invention;
Figure 1B shows in more detail the internal structure of the conversational service component of Figure 1;
Figure 2 shows the internal structure of a role component of the conversational service component of Figure 1;
Figure 3A shows a state transition diagram of an Initiator role component according to the invention;
Figure 3B shows a state transition diagram of a respondent role component according to the invention;
Figure 4 shows generic agent roles in a multi-agent architecture according to the invention;
Figure 5 shows the structure of the co-ordination engine and the process for downloading, installation, and execution of an Initiator role component according to the invention;
Figure 6 shows the process for loading an instance of a conversational service component according to the invention;
Figure 7 shows an embodiment of the invention where a user is requesting services on demand from a server;
Figure 8A shows an embodiment of the invention in which information service and provider agents interact via a mediator agent; and
Figure 8B shows an embodiment of the invention comprising a multi-agent system for airline-ticketing.

There follows a detailed description of the preferred embodiments of the invention, which include a description of the best mode of the invention as currently contemplated by the inventors. Even where not explicitly described, it will be apparent to those skilled in the art that certain features of the invention can be replaced by their known equivalents, and the scope of the invention is intended to encompass such equivalents where appropriate.

Overview

A multi-agent system (MAS) architecture according to the invention allows plug-and-play of service components (C-COMs) into software agents to facilitate inter-agent interactions. The C-COM provides a "conversational" facilitator for an inter-agent interaction. The C-COM software is arranged to enable required agent interactions (i.e. a request and response) to occur for a given service when the C-COM software is loaded appropriately onto the agents participating in the interaction.

The C-COM structurally comprises a plurality of role components for the interaction (where the number of role components provided is determined by the number of roles needed for the interactions). All the service interactions are performed via messages which can be interpreted and composed by the role components. As the role components are aware of the content language, ontology, and interaction protocol used for a given service, an agent can participate (as a client, a server or both) just by installing one or more C-COMs for a given service, even if the agent has no awareness of the required content language or interaction protocol.

The generic roles for the C-COMs are 'Initiator' and 'Respondent'. These roles can be specialized into more roles according to the application requirements. The 'Initiator' role component is installed in a client agent and is required to obtain the service result from a server agent. The 'Respondent' role component is installed in a server agent in order to provide services to other client agents. The client agent and server agents interact via the messages which are generated and interpreted by their respective role components according to a pre-defined content language, ontology, and interaction protocol for the service in question.

One embodiment of the invention provides a mediator agent for a multi-agent architecture (MAS) which provides the role components to the software agents dynamically, i.e., on the fly. For example, a mediator agent may provide an 'Initiator' role component to a client agent dynamically whilst the client agent is already running in the MAS. As a result, the client agent does not need the 'Initiator' role component a priori, instead, the client agent can request a service and receive it without prior knowledge of the content language or interaction protocol required by the server providing the requested service..

The appropriate mediator agent for a software agent in a MAS is identified by determining whether the mediator agent has a suitable Initiator role component for the software agent's desired inter-agent interaction. This is achieved through the use of a service component description of the service component which contains a plurality of role components, including the necessary Initiator and Respondent role components. The mediator agent then provides the role components dynamically and these are loaded on the software agent whilst the agent is running in the MAS. Once an Initiator role component is loaded from a mediator agent, the client agent is directly able to request a service from the available service agent. Suitable server agents are identified by the downloaded Initiator role component when the client agent executes it to get a service result.

Advantageously, the resulting MAS has a very loosely coupled architecture which allows agents to join, disengage and/or rejoin with minimal impact, just by installing one or more C-COMs.

A service component (C-COM) for plug and play in a software agent in a multi-agent system (MAS) is therefore a software component which is used as the principle means of interaction between software agents, and which can, in preferred embodiments of the invention, be installed and executed by a software agent dynamically.

<u>C-COM architecture</u>

Referring now to Figure 1A, the structure of a service component (C-COM) for a software agent according to a first embodiment of the invention is shown schematically. In Figure 1, C-COM 1 comprises four role components 2,3,4,5 . Each role component 2, 3,4, 5 can be plugged dynamically into an agent. Four agents are shown schematically in Figure 1, as agents 6, 7, 8, 9.

From an agent's point of view, each role component is a black box which hides the interaction protocol 10 with other role components and just shows an interface which specifies the input and output data needed to execute the role component. As shown in Figure 1, a role component 2,3,4,5 is plugged into an agent 6,7,8,9 respectively when that agent 6,7,8,9 participates in an interaction. However, more than one role component can be plugged into an agent. An agent which installs a role component is called a Master Agent of the role component.

The C-COM is an implementation of an interaction pattern which is a template of a frequently used interaction scenario amongst roles (for example, the interaction pattern for an auction can be described as follows. First, the seller advertise something to sell, second, the buyers send their bids to the seller, finally, the seller sells the product to a buyer who has sent the best bid before the closure of the auction). The C-COM is formed as a re-usable software component based on this interaction pattern and is generated in a form which can be dynamically installed on and executed by other software agents. A conversational component (C-COM) is therefore an implementation of a reusable interaction pattern which produces a service via interactions among role components within the conversational component.

The interactions between role components of the C-COM are controlled by an Interaction Protocol (IP). Each role component performs actions in each stage of the Interaction Protocol to produce the required service. A role component is defined as a finite-state-machine that performs actions according to its current state to produce the required service. A role component reveals a set of interfaces to its user. The structure of a role component is discussed in more detail later with reference to Figures 1B and 2 of the accompanying drawings.

Accordingly, the CCOM can be denoted as follows:

$$ccom = <IP, Cs>$$

where IP is an Interaction Protocol, as defined below, and Cs is a set of finite state machine-like components having one or more roles defined in the IP.

The term Interaction Protocol (IP) describes what sequences of which messages are permissible among a given set of roles, for example, see the specifications of standard interaction protocols (by FIPA) from http://www.fipa.org/repository/ips.html.

The multi-agent system architecture of the invention is arranged to enable plug-and-play of conversational components (CCOMs) to allow software agents to interact with unknown content languages and interaction protocols. This requires the software agents to have the same understanding on the ontology used within messages. As described herein above, the conversational components (CCOMs) comprise software components which enable the required agent interactions (i.e., request and response) to occur during an information trade. The CCOM consists of two or more role components (according to the roles needed for the interactions) and all the interactions are performed via messages which can be interpreted and composed by the role components.

The generic role components are the Initiator and Responder role components. The Initiator and Responder role components are aware of the content language, ontology, and IP used for any given information trade. From an agent's point of view, the Initiator role component is a black box which hides the interaction process with its designed Respondent role components from its master agent (the agent which installs the role component) and only exposes an interface which specifies the input and output data needed to execute the role component, and more than one role component can be plugged into an agent concurrently.

The invention thus enables software agents such as an information provider agent to participate in an existing agent society by providing its Initiator role components to information consumer agents via a public mediator facility. An information consumer agent is able to interact with the new information provider agent by installing the Initiator role component for a given information trade, even if it doesn't recognize the required content language or interaction protocol. The multi-agent system architecture (CCoMaa) defines the roles of participating agents and the procedures which they should comply with in order to install C-COMs and to be capable of interacting with the corresponding agents during the information trade.

Referring now to Figure 2, an internal structure of a role component is shown schematically. Each role component consists of four main modules: a Protocol Manager 11, an Interface (either an Inner interface 12a or an outer interface 12b although both are shown in Fig. 2) , an Action pool 13 , and Message Handler 14.

The Protocol Manager 11 controls all the actions of a role component according to the interaction protocol employed by the role component and the nature of the role component in the interaction protocol (see Figure 1). The Protocol Manager 11 interacts with the Message Handler 14 to send and receive messages. The Message Handler 14 filters messages which are sent to the role component from the message queue of its Master Agent.

In the context of the invention, both interfaces 12a,12b are defined as a set of method signatures that specify the method name, input arguments, and output of the method (c.f. the definition of 'Interface' in Java). When a role component interacts with its Master Agent, the Master Agent installs the implementation of the appropriate interface 12a or 12b, in accordance with the role of role component 2.

Inner interface 12a defines a trigger method (in that a call to the method by a service consumer activates the function of the whole C-COM) which is called by a service consumer to get a service result from the role component 2. The role component 2 implements the inner interface 11a to produce the required service result.

Outer interface 12b defines methods which are called by the role component 2 to get application specific input. The agent which installs the role component 2 should provide an implementation of the outer interface 12b. Then, the role component 2 interacts with the implementation to produce the required service result. The outer interface 12b enables the customisation of the role component 2 for different application requirements. For instance, a Respondent role component can be reused in different applications by providing different implementation of outer interface 12b. The communication between the role component and application specific implementation via outer interface is controlled via Coordination Controller 51 in

Figure 5. More specifically, all the request for application specific input value is to through the outer interface to the Coordination Controller which is then responsible for getting the required value from a human user defined implementation or by executing another CCOM.

An Initiator role component is a role component which has an inner interface 12a and an outer interface 12b. A Respondent role component is a role component which has an outer interface 12b, but not an inner interface 12a.

The Initiator role component is activated when the trigger method defined by the inner interface 12a is called by a service consumer. The activated Initiator role component initiates interactions with Respondent role components within a C-COM to produce a service result. Figure 1B describes the internal structure of a C-COM in more detail showing the Initiator and Responder role components, and is described in more detail later. Details of the interaction between state transitions of an Initiator role component are shown in Figure 3A and discussed in more detail later.

The Respondent role component is activated when a triggering message arrives from an Initiator role component. The structure of a Respondent role component is shown in more detail in Fig. 1B. Details of the state transitions of a Respondent role component are shown in Figure 3B and discussed later. In this context, a triggering message is defined as the first message for the role the Respondent role component is responsible for, in the Interaction Protocol 10 employed in the C-COM.

Figure 2 schematically shows state transitions within a role component. When a role component is installed into a Master Agent, the component has "Ready" state. When any methods in Inner Interface/Outer Interface or Actions are executed by Protocol Manager 11, the component transit to next states ("State 2", "State 3", and finally "Completed"). Once the component reaches to "Completed" state, it automatically resets its state to "Ready" waiting another request from a service consumer (for the Initiator role component) or the Initiator role component (for the Respondent role component).

The Initiator and Respondent role components are generic in that they can be specialised for more specific role components according to the requirements of the target C-COM. A Master Agent can handle multiple trigger messages concurrently by installing multiple Respondent role components.

Fig. 1B shows in more detail the interaction between the Initiator and Respondent role components within the internal structure of a C-COM according to the invention. The interaction protocol (IP) defines the sequence of asynchronous messages sent between the role components, and the role components perform the actions necessary at each stage of the interaction protocol to achieve the service goal.

As shown schematically, in Figure 1B, the Initiator role component starts an interaction by sending a message and the Respondent role component is activated when it receives a message from the Initiator role component. These two generic role components can be specialised according to the requirements of target applications.

In Figure 1B, each role component consists of an Interaction Protocol Scheduler (IPS), a Message Handler (MH), an Action Pool (AP) and one or more Interfaces. Each role component is in effect a Finite State Machine, driven by internal state changes, and has a different set of internal states according to the role the component plays in the interaction protocol employed for a given C-COM. The IPS schedules and executes all the actions stored in the AP of a role component according to internal state changes.

For this purpose, each role component maintains an Interaction State which is managed by the Interaction State Manager (ISM). The MH is responsible for validating outgoing messages and interpreting incoming messages. The role component provides a number of interfaces for customisation purposes. In this context, an interface is defined as a set of method signatures. An interface must be provided with an implementation of the method signatures to be executed at run time.

An Initiator role component has two kinds of interface: External and Internal (EI and II respectively). An EI defines the signature of a trigger method (which triggers the execution of the entire C-COM), i.e., the input data and the service result which is returned to its master agent. Calling the trigger method in the EI activates the Initiator role component which activates all its other Respondent role components in order. An II is an information channel from a master agent to a role component. A role component is able to ask its master agent to provide ontology items that are necessary to create a message. For example, if a Respondent needs access to a knowledge source to get information to populate a response message, its master agent should provide the Respondent role component with the requested information. The same is necessary for an Initiator role component.

Figures 3A and 3B show the state transition diagrams which show the internal functionality of the Initiator and Respondent role components respectively.

In Figure 3A, an installed component is initially in an "Idle" state 20. After a service request has been issued, the component is in the "Schedule Next Behaviour" state 21. The behaviour execution request then causes the component to execute the behaviour (state 22) and once this is finished the component returns to its schedule next behaviour state 21. When a new interaction is required the component prepares request message 23. Once the request message has been sent, the component is in a wait response message state 24. Once the response message has been received, the component is in a handle response message state 35. The response message is then interpreted by the software component which then enters its "Schedule Next Behaviour" state 21, before delivering the service result and returning to the idle state 20.

In Figure 3B, an installed component is initially in the "Idle" state 26. After a service request has been issued, the component is in the "Schedule Next Behaviour" state 27. The behaviour execution request then causes the component to execute the behaviour (state 28) and once this is finished the component returns to the "Schedule Next Behaviour" state 27. When a new interaction is required the component enters the "Activate Agent Interface" state 29. Once the agent response is received, the component again returns to the "Schedule Next Behaviour" state 27. The response message is then prepared and the agent enters the "Send Response Message" state 30. Once the message has been sent, the agent returns to the "Schedule Next Behaviour" state 27 before returning to "Idle" state 26.

A multi-agent architecture (CCoMaa) according to the invention enables agents to interact with each other through C-COMs. This allows existing agents to interact with new service-providing agents by dynamically installing and executing Initiator role components which are provided by the new service providing agents. To facilitate this dynamic configuration change management, the CCoMaa needs special agents having predefined roles. Fig. 4 shows schematically some generic agent roles in the CCoMaa and their interactions.

In the multi-agent architecture (MAS) according to the invention, an agent may have one of a plurality of roles. For example, as Fig. 4 shows schematically, the agent may be a service provider agent (SPA) 41, a service mediator agent 42, and a service consumer agent (SCA) 43. It will be appreciated by those skilled in the art that the term "SPA" is equivalent in use to the term "server agent" and that the term "SCA" is equivalent to "client agent". An information provider agent (IPA) is an example of a SPA, and a information consumer agent (ICA) is an example of an SCA.

The SPA 41 registers the service description, component descriptions, and executable Initiator role components (actions "A" shown in Fig. 4). The SPA 41 registers its service description to the service mediator agent 42 by a process which differs from the one defined by FIPA.

More specifically, according to the invention, the SPA 41 registers an executable Initiator role component, a component description (which is used by the SCA for installation and execution of the executable Initiator role component), and a service description. Advantageously, as the Initiator role component generates and interprets all messages which will be exchanged by the SPA 41 with a SCA 43, there is no need for a SCA 43 to have knowledge a priori to requesting a service of the requirements imposed by the SPA 41.

The service mediator agent SMA 42 according to the invention is responsible for maintaining the service registry and Initiator library that contains Initiator role components (actions "B" in Fig. 4). The service registry plays the role as defined in FIPA specifications or other similar systems. The Initiator library maintains the pair of component description and the executable Initiator role component. The component description specifies the information needed to install and execute the executable Initiator role component.

The difference between the service registration process in this invention and that of FIPA can be explained in detail as follows in which a specific embodiment of the invention relating to an airline ticketing scenario is described. This embodiment is described in more detail later with reference to Figure 8 of the accompanying drawings..

FIPA service registration

Firstly, in the FIPA specifications, a FIPA SPA registers its description to a Directory Facilitator (DF) agent as per the following example;

```
<DFAgentDescription>
        <Name>
        <Agent-Identifier      name="routeplanner@foo.com",
address="iiop://foo.com/acc" />
        </Name>
        <Protocol name="AdHoc-Protocol" />
        <Ontology name="Travel" />
        <Language  name="FIPA-SL0" />
        <Language  name="KIF" />
        <ServiceDescription>
             <Name> bookFlight </Name>
             <Type> BookingService </Type>
             <Ontology> Travel </Ontology>
             <Protocol> FIPA-Request </Protocol>
```

```
            <Property name=domain, value=international />
        </ServiceDescription>
</DFAgentDescription>
```

The DF agent stores the DF agent description in its local table. When a SCA sends a request message to the DF agent to find a SPA for a given service, the DF agent returns the names of any suitable SPAs it finds to the SCA. Once the SCA receives the name of one or more SPAs, it starts an interaction with those SPAs employing the ontology, language, and interaction protocol specified in the service description. The format of the query message the SCA sends to the DF agent is as follows:

```
(request
        :sender (agent-identifier :name SCA@foo.com :address iiop://foo.com/acc)
        :receiver (agent-identifier :name Mediator@foo.com :address iiop://foo.com/md)
        :language FIPA-SL0
        :protocol FIPA-Request
        :ontology CCOM-Management
        :content
            (action (agent-identifier :name Mediator@foo.com :address iiop://foo.com/md)
                (search
                    (ccom-agent-description
                        :ontology (set Travel)
                        :language (set FIPA-SL0 KIF)
                        :services (set
                            (service-description
                                :name bookFlight
                                :type BookingService
                                :ontology Travel
                                :language SL0
                                :protocol FIPA-Request
                                :properties (set
```

(property :name domain :value international))))))))

According to the invention, however, the above message is used only by a Mediator Agent to find appropriate SPAs and returns the list of names of suitable SPAs (with each entry in the form of "<Agent-Identifier name="routeplanner@foo.com", address="iiop://foo.com/acc" /> ") to the SCA.

Service Registration using Role Components

On the other hand, the SPA of the invention registers the following form of description to a service mediator agent (SMA) . The example is provided in pseudo code and comprises a sample CCoMaa agent description which is used to register a service and Initiator component by an SPA comprising an IPA (see also the description referring to in Figures 8A and 8B which relate to an embodiment of the invention concerning airline ticketing information):

```
<CCOMAgentDescription>
        <Name>
              <Agent-Identifier name="dummy@foo.com",
address="iiop://foo.com/acc" />
        </Name>
        <Protocol name="AdHoc-Protocol" />
        <Ontology name="Travel" />
        <Language  name="FIPA-SLO" />
        <Language  name="KIF" />
        <ServiceDescription>
              <Name> bookFlight </Name>
              <Type> BookingService </Type>
              <Ontology> Travel </Ontology>
              <Property name=domain, value=international/>
        </ServiceDescription>
        <ComponentDescription>
```

```
            <package name="com.travelagent.booking"
                  mainclass="Flight_initiation.class" />
            <MinJVM> JDK1.1.x </MinJVM>
            <InnerInterface>
                  <Method name=getTicket>
                        <Input          order=1,          name="Route",
type=Travel.Booking.Route />
                        <Output type=Travel.Booking.TicketList />
                  </Method>
            </InnerInterface>
      </ComponentDescription>
</CCOMAgentDescription>
```

The CCoMaa agent description thus consists of a service description and a component description. The service description consists of name, type, and properties fields. The component description has three sub-descriptions: Interface, File and graphical user interface (GUI) descriptions.

The Interface description specifies the trigger method and ontology items used as input and output of the trigger method. The File description is used by the ICA to install the downloaded Initiator component in the local device on which the agent is running. Finally, the GUI description is optional. A SPA can provide a SCA with a GUI component which can be used to retrieve additional ontology items (which might not be known to the SCAs) used by the Initiator role component to pass to the Respondent role component.

Advantageously, once the service provider has registered its service(s) to an SMA, any SCA 43 can contact the service mediator agent 42 to get contact information for the SPAs 41. To get the contact information, the SCA 43 sends a message that contains a service description to the service mediator agent 42 (action "C" in Fig. 4). The content of the message can be as per the following example (again given in the context of the SCA comprising an ICA, and the SPA comprising an IPA, see Figure 8B for more details).

```
(request
        :sender (agent-identifier :name SCA@foo.com :address iiop://foo.com/acc)
        :receiver (agent-identifier :name Mediator@foo.com :address iiop://foo.com/md)
        :ontology CCOM-Management
        :language SL0
        :protocol FIPA-Request
        :content
             (action (agent-identifier :name Mediator@foo.com :address iiop://foo.com/md)
                    (search
                           (ccom-agent-description
                              :services (set
                                 (service-description
                                    :name bookFlight
                                    :type BookingService
                                    :properties (set
                                       (property :name domain :value international))))
                              :component (set
                                 (component-description
                                  :min-jvm jdk1.2.x
                                  :inner-interface (set (method :input Travel.Booking.Route))))))
```

This request message is also different from that of FIPA. Note that the SCA doesn't have to specify language and protocol in a service description used for service acquisition. The service mediator agent 42 then tries to match the service description and component description with one provided by SPA 41. If a matching agent description is found, the mediator agent 42 returns the agent description of the SPA 41 and an executable Initiator role component to the SCA 43 (actions "D" in Figure 4). If any contact information is returned by the service mediator agent 42, the SCA 41 installs the executable Initiator role component into its Initiator library, based on the component description (action "E" in Figure 4) and executes the Initiator role component to get the service result from the SPA (actions "F" and "G" in Figure 4).

Dynamic download, installation and execution of C-COM

The SCA 41 shown in Fig. 4 is equipped with a co-ordination engine which is responsible for downloading, installing, and executing an Initiator role component. Referring now to Figure 5, the structure of the co-ordination engine and the process for downloading, installation, and execution of an Initiator role component is shown schematically. The co-ordination engine forms part of an agent internal architecture arranged to enable the dynamic installation and execution of role components. The agent internal architecture comprises: a Co-ordinator controller, a Load manager, a Component installer; and a Package manager.

In Figure 5, co-ordination controller 51 activates a load manager module 52 when an executable Initiator role component instance is requested to perform a required service by another component from the SCA 43 (see Fig. 4). If the Initiator role component has already been downloaded before, the load manager 52 module may (depending upon the requirements of the application) perform a version check to ensure that the latest version is installed. The versioning process is determined by the version control scheme currently in place. Otherwise the load manager 52 module will instantiate the Initiator role component and pass the instantiated object back to the co-ordination controller 51 where it will be executed. If there is no locally installed copy of the Initiator role component the load manager module 52 forwards a request to the component installer module 53, which will attempt to locate a service mediator agent 42 (see Fig. 4) that contains the required Initiator role component.

There are a variety of mechanisms that can be used to discover the available service mediator agents within the network. For example, if the application is running on a FIPA compliant agent platform, the directory facilitator can be used. When the component installer module 53 locates a C-COM that matches the given requirements, the component installer 53 downloads the Initiator role component of the C-COM.

The component installer 53 then returns the results of its search to the load manager 51. If successful the request will contain the Initiator portion of the Initiator role component. The load manager 51 then stores a copy of the Initiator role component within the C-COM library 55, and requests that the Package manager 54 installs the Initiator role component so that it can be executed. The Package manager 54 then loads the files into the file system, and control will return to the load manager 51. Once the Initiator has been downloaded and stored, the load manager 51 records the details of the process so that in future the Initiator role component does not need to be downloaded again unless a new version becomes available. The result of the interaction with the SPA 41 is returned to the requester of the service result.

Fig. 6 shows steps in a method for loading an instance of a C-COM by the collaboration of Load Manager 52, Component Installer 53, and Package Manager 54 in Figure 5. First, Load Manager 52 asks Package Manager 54 if there are any Initiator-role components available for a service request. If there exists any Initiator-role components 61, then the Load Manager 52 performs a version check 63 by sending a command to the Component Installer 53. The Component Installer 53 contacts the Mediator Agent 42 to get the latest version number for the corresponding Initiator role components. If the local Initiator role component is the latest version, the Load Manager 52 instantiates the role component and returns the instance to the Coordination Controller 51 and finishes the process 69.

If there is no role component managed by the Package Manager 54, the Load Manager 52 asks the Component Installer 53 to download any Initiator role component. Then the Component Installer 53, first, locates a Mediator Agent 62. If no Mediator Agent is found 65, the process is finished. Otherwise, the Component Installer 53 queries the found Mediator Agent 42 to get any appropriate Initiator role components 64. If found, the Component Installer 53 downloads the found Initiator role components 66 and passes them to the Package Manager 54 to store in its local component store for later use 68, and finally instantiates the Initiator role component 67 and finishes the process 69.

Advantageously, one embodiment of the invention can be used to implement a dynamic version management system in which service providers can upgrade services without affecting their customers by providing upgraded Initiator role components on the mediator agent. The SCA needs to only compare versions of the local Initiator and remote Initiator when it contacts the mediator agent. The version upgrade is then done automatically before actual interaction with the SPA is performed by the SCA, in a manner which is hidden from any human users.

Advantageously, the invention enables a user to have improved access on-demand to the latest version of an application provided by a remote service over a data communications network. For example, referring now to Fig. 7 of the accompanying drawings, a schematic diagram is indicating how a first user 70 can be provided with access on-demand to remote services provided by service provider 71 via a brokerage 72. The application used by user 70 provides a client agent which interfaces with an Initiator role component 73 provided by a suitable mediator agent provided by the brokerage 72.

Initiator role component 73 interacts with a Respondent role component 74 set up by the mediator agent, and is able to interface with a SPA of the service provider 71. The Initiator and respondent roles form part of a C-COM arranged to facilitate the software agents' interaction with each other in the manner described hereinabove with reference to the other embodiments of the invention.

The first user 70 is able to access the latest version of the service they have requested from the service provider without having any previous requirement to install any components specific to that version, as any required conversational components (i.e. Initiator and responder) will be installed as appropriate dynamically into the brokerage agent. Moreover, once the client agent of a second user 75 is provided with access to an appropriate Initiator role component 76, the client agent is able to interact directly with Respondent role component 74 as this is already associated with that type of Initiator role component.

In this way, the service provider and user agents can interact regardless of previous awareness of the interaction protocol or language required by the particular version of the application which the user wishes to access.

Advantageously, another embodiment of the invention can be used to implement a web service portal where new web service providers can be registered without affecting other existing agents.

Figure 8A shows an embodiment of the invention in which an information trade occurs between a client agent and a service provider agent in a multi-agent system architecture (CCoMaa) according to the invention. In Figure 8A, the three generic agent roles are the Information Consumer, the Information Provider, and the Information Mediator. In the CCoMaa shown in Figure 8A, an Information Mediator Agent mediates all interaction between an Information Provider Agent (IPA) and an Information Consumer Agent (ICA).

As shown in Figure 8A, an IPA first registers its service and component description to an IMA (step A). The IPA registers a service description (as required by FIFA) and additionally an executable Initiator component and a component description, which is used by the ICA during the installation and execution of the executable Initiator role component.

The IMA is responsible for maintaining the service registry and Initiator library that contains Initiator role components (B). The service registry maintains all the registered service descriptions as defined in FIPA specifications. The Initiator library maintains the pair of role component description and the executable Initiator role component. The role component description specifies the minimum execution environment needed to install and execute the Initiator role component. This includes the required runtime environment (e.g. a JVM supporting the CLDC specification) and required computing resources (e.g. 20k storage space, 160x160 screen resolution, etc).

Once the IPA has registered its service(s), any ICAs can contact the IMA to retrieve the contact information on any registered IPAs. To retrieve the contact information, an ICA needs to send a query message containing a service description (C). The IMA then tries to match the service description with one provided by an IPA. If a matching service description is found, the IMA returns a tuple containing a component description of the IPA and an executable Initiator component to the ICA (D). If any contact information is returned by the IMA, the ICA installs the executable Initiator component into its Initiator library, based on the component description (E) and executes the Initiator component to get the service result from the IPA (F), (G).

Figure 8B shows schematically a simplified embodiment of the invention based on an airline-ticketing scenario. This scenario is based on a ticket consumer agent (TCA) which is assisting its owner in the purchase of an airfare from one of two available ticket provider agents (TPA).

In the embodiment shown in Figure 8B, each TPA represents a different airline carrier and although they both adhere to the 'official' airline booking specification, they also utilize different IPs as a way of enhancing and differentiating their services from competitors. TPA (A) provides an additional airport transfer service that allows users to book a variety of transport modes to and from the airport. In contrast TPA (B) provides two additional services: 1) **seat booking**, which allows users to book their preferred seating position, and 2) **food selection**, which allows a user to choose their preferred meal from a selection of available meals.

Within this example the following assumptions are made:

1) The TCA has on previous occasions interacted with TPA (A) and as such already has an installed copy of the appropriate C-COM.
2) The TCA contains an implementation of a graphical user interface (GUI) which can be used to extract information from the user regarding the type of airfare they wish to purchase. This GUI captures the values defined within the official airline booking specification. Values such as origin, destination, airfare type and price are captured.

Firstly, the TCA enables the user to specify via the TCA GUI which type of airfare they would like to purchase. Once this information is obtained the TCA requests a 'Travel.Booking.Ticket' object from the local co-ordination engine, which responds by loading and executing the Initiator role component from TPA (A). A three-phase IP is used during the execution of this Initiator role component:

Phase one: The Initiator role component takes the input provided within its trigger method and sends a request message (1) to the corresponding Respondent role component of TPA (A). If any available airfares match the provided input they are included within the response message (2) sent back to the Initiator role component.

Phase two: The Initiator role component's GUI will then show the list of matched airfares. Once the user selects an airfare, a request message (3) is sent back to the Respondent role component indicating which airfare the user wishes to book. The Respondent role component does not process the booking itself, rather it uses the services of a finance provider agent (FPA). The Respondent role component asks its local co-ordination engine for a 'Travel.Booking.TicketReciept' object which responds by loading and executing (4) an Initiator role component. This Initiator role component sends a request message (5) to the Respondent role component of the FPA, which performs the booking process. Upon completion of the booking process a response message (6) is sent back to the Initiator role component of IPA (A) containing the 'Travel.Booking.TicketRecipent' object. This object is then given (7) to the Respondent role component of TPA (A) by the co-ordination engine. To complete phase two of the IP the Respondent role component sends a confirmation message (8) back to the Initiator role component of the TPA.

Phase three: The Initiator role component's GUI then allows the user to book transport to the airport if they wish. If the user chooses to book transport, then the Initiator role component sends a request (9) message to the Respondent role component. A response message (10) is then sent back to the Initiator role component.

Now consider the case where IPA (A) is unable to fulfil the service request. This is shown in Figure 8. In such a situation, the ICA is forced to locate another IPA which is achieved by contacting the nearest IMA. In this scenario there is only one other IPA, (B). Once the ICA locates IPA (B) via the IMA, it will complete the C-COM installation steps (described hereinabove with reference to Figure 4). The Initiator role component of IPA (B) employs a different IP to the previous Initiator role component of IPA (A). A four-phase IP is used by the Initiator role component of IPA (B). Phases one and two are similar to the previously discussed IP with the exception that the Respondent role component of IPA (B) internally processes the airfare booking. Phases three and four are described below:

Phase three: Once the user has booked their airfare the Initiator role component's GUI allows the user to select their preferred seating position. Once selected, a request message (5) is sent to the Respondent role component of TPA (B), which is followed by a response message (6).

Phase four: The final phase of the IP involves the selection of food for the flight. The Initiator role component's GUI allows the user to indicate their food choice. A request message (7) is sent to the Respondent role component of TPA (B). The Respondent role component takes the users food choice and requests a 'Travel.Booking.FoodReceipt' object from the local co-ordination engine. The co-ordination engine loads (8) an Initiator role component to fulfil this request. The Initiator role component then sends a request message (9) to the corresponding Respondent role component located in a food provider agent (FPA). The Respondent role component responds (10), and the Initiator role component of TPA (B) passes the request object (11) back to the co-ordination engine. The Respondent role component then sends a confirmation message (12) back to the Initiator role component of TCA.

The component-based multi-agent architecture thus enables open and flexible information exchange among multiple agents. The C-COM plug & play software component comprises two or more role components which abstract and hide all the details of interactions among the participating roles. The CCoMaa is a multi-agent architecture where participating agents can interact with each other via a C-COM.

The CCoMaa enables an agent to participate in new services that have been added to an existing agent society even if the agent doesn't recognize the interaction protocol (IP) or language used by the new services. Simply by installing the appropriate role component an agent is able to take advantages of these new services.

The invention requires participating agents in the CCoMaa to have the same understanding of the ontology items used for their interactions. If an IPA requires an additional unknown ontology item from an ICA, the IPA is forced to provide a GUI component that is then used to retrieve the unknown ontology items from the ICA's human user. It is possible to increase an ICA's autonomy by providing instead an ontology derivation rule which guides an ICA as to how the unknown ontology item can be derived from known ontology items.

The use of C-COM can prevent an ICA from learning the social interactions used with the other participating agents as a C-COM hides all the interaction details. An extended component description can be provided to resolve this by detailing the IP employed by the C-COM.

It will be appreciated by those skilled in the art that many aspects of the invention can be implemented in either software and/or hardware and that the spirit of the invention is intended to cover embodiments in any combination of software and/or hardware as appropriate, and that software elements of the invention can be provided by a computer product which may comprise a signal communicated over a communications network, for example, as a downloadable suite of one or more computer programs, and/or as a suite of one or more computer programs provided on a computer readable data carrier, which are able to execute the invention when loaded and run on an appropriate device. The invention can also be provided by a carrier having a computer program stored thereon, the computer program being executable on a terminal so as to cause the terminal to operate the invention.

Any pseudo-code described herein is provided as is and without any guarantee as to its completeness or accuracy.

CLAIMS

1. A service component for a software agent, the service component being arranged to enable a client agent to interact with a server agent when requesting a service, the service component comprising:

a plurality of role components arranged to perform a service interaction between the client agent and the server agent, the role components being arranged to be loaded onto said client and server agents as appropriate for the interaction and when loaded being arranged to provide the client and server agents with information on the interaction requirements to enable the requested service to be provided.

2. A service component as claimed in claim 1, wherein at least one of said role components comprises an Initiator role component provided by a service provider agent to a service consumer agent.

3. A service component as claimed in either claim 1 or claim 2, wherein at least one of said role component is attached to a component description, the component description including details of the minimum client platform capability of the client agent and the interfaces used by the client agent to interact with the role component.

4. A service component as claimed in any previous claim, wherein at least one of said role components comprises an Initiator role component which can control its state.

5. A service component as claimed in any previous claim, wherein at least one of said role components comprises an Initiator role component which can be reused for multiple requests.

6. A service component as claimed in any preceding claim, wherein each of the plurality of role components is distributed by a mediator agent.

7. A service component as claimed in claim 6, wherein the mediator agent provides each of the plurality of role components dynamically to the client and server agents.

8. A service component as claimed in any one of claims 5 or 6, wherein the mediator agent selects one of said plurality of role components as suitable for distribution by using a service description and component description of the role component.

9. A service component as claimed in any preceding claim, wherein one of said plurality of role components is an Initiator role component provided dynamically to the client agent whilst the client agent is running.

10. A service component as claimed in any preceding claim, wherein one of said plurality of role components is a Respondent role component provided dynamically to the server agent whilst the server agent is running.

11. A service component for a software agent, the service component being arranged to enable a user to request a service using a client agent, the client agent arranged to interact with a server agent when requesting the service, the service component comprising:

a plurality of role components arranged to perform a service interaction between the client agent and the server agent, the role components providing the client and server agents with information on the interaction requirements to enable the requested service to be provided, wherein the service component is generic to the client and server agents and is dynamically installed into at least one of the client and server agents when these agents are already running.

12. A multi-agent service architecture having a service component arranged to enable a client agent to request a service from a service agent, the architecture including:

a mediator agent arranged to provide a role component to the client agent, wherein once the role component is loaded on the client agent, the client agent is provided with information which enables the service to be provided by the service agent.

13. An agent internal architecture for dynamically installing and executing role components, the architecture comprising:

a Co-ordinator controller;

a Load manager;

a Component installer; and

a Package manager.

14. A method of providing a user with access on demand to a remote service, the method comprising the steps of:

generating a client agent for the user to request the service from a server agent;

providing the client agent with at least one service component arranged to modify the client agent to enable the client agent to interact with the server agent when requesting the service;

forwarding the modified client agent to a broker to enable the server agent and modified client agent to interact; and

responding to the client agent's request to provide the requested service,

wherein the service component provided comprises:

a plurality of role components arranged to perform service interactions between the client agent and the server agent, the role components providing the client and server agents with information on the interaction requirements to enable the requested service to be provided.

15. A method of providing a user with access on demand to a remote service as claimed in claim 14, wherein the plurality of role components are provided by a mediator agent.

16. A method of enabling a software agent to participate in an inter-agent interaction in a multi-agent system architecture, the method comprising the steps of:

determining at least one of a plurality of role components to be used by a service component of said software agent when required for participation in the inter-agent interaction;

identifying a mediator agent in the multi-agent system which is capable of providing at least one role component required by the software agent for participation in the inter-agent interaction, the mediator being identified by means of a service component description as having a suitable role component for the service component;

dynamically installing the at least one role component provided by the mediator agent on the software agent; and

loading the at least one role component on the software agent to enable the software agent to participate in the inter-agent interaction.

17. A method as claimed in claim 16, wherein the software agent is a client agent, and at least one role component provided by the mediator agent is an Initiator role component, and the inter-agent interaction comprises a request for a service by the client agent from a server agent.

18. A method of enabling a software agent to manage one or more role components, the method comprising the steps of:

determining whether if one or more role components are stored in a downloaded form in a local component storage element; and

if a downloaded role component is found in a local component storage element, determining if the downloaded role component is an Initiator role component; and

if the downloaded role component is an Initiator role component,

performing a version check of the downloaded Initiator role component; and

if the downloaded role component is not an Initiator role component, locating a Mediator agent having at least one Initiator role component; downloading at least one Initiator role component from the Mediator Agent;

packaging at least one downloaded Initiator role component into local component storage; and

instantiating the downloaded Initiator role component.

19. A computer product comprising a suite of one or more computer programs provided on a computer readable data carrier, the one or more computer programs being arranged to implement any one of the methods of claims 14 to 18.

20. A signal conveying a suite of one or more computer programs over a communications network, the suite of one or more computer programs being arranged when executable to implement any one of the methods of claims 14 to 18.

ABSTRACT

A service component enables client/server interactions even when information on the content language and/or interaction protocol required for the service the client agent has requested from the service agent is not known *a priori*. The service component has a generic structure comprising a plurality of role components which perform the service interaction between the client agent and the server agent and which provide sufficient information on the interaction requirements to enable the requested service to be provided.

Figure (4)